AQUAS: A Project to Bridge the Gaps between Safety and Security Processes

John Favaro, Silvia Mazzini
Intecs SpA, Pisa, Italy; email: {John.Favaro,Silvia.Mazzini}@intecs.it

Peter Popov, Lorenzo Strigini
Centre for Software Reliability, City, University of London, U.K.; email: {ptp,strigini}@csr.city.ac.uk

1 Introduction

We report on an approach to the management of the interplay between the safety and security processes, currently studied in a recently started collaborative European project, AQUAS (Aggregated Quality Assurance for Systems, http://aquas-project.eu/). AQUAS is experimenting with co-ordinating these processes through *"interaction points"*, which will be introduced below, via a set of case studies or "demonstrators". It is motivated by the problems found by industry in combining in a cost-effective way the tasks of ensuring satisfaction of various non-functional requirements (where "ensuring" means "achieving and demonstrating").

Most such problems have been reported with the task of ensuring both safety and security in embedded systems. Companies with established processes for ensuring safety would import processes for ensuring security as well, but problems may arise because on the one hand, the two need to be considered together (e.g. because security violations affect safety, and because design trade-offs may arise between these two sets of goals), but on the other hand, they are the preserves of different technical cultures with their own languages, habitual assumptions in their analyses, etc. It is sometimes said, deprecatingly, that these specialists of different cultures work in "silos", with information flowing vertically within a specialism but not across specialisms. As the SAE J3061 Cybersecurity Guidebook has noted: “A tightly integrated process for Cybersecurity and safety has the advantage of a common resource set, thus, requiring fewer additional resources. However, since both activities require different technical expertise and both activities are resource intensive, it may not be feasible to expect a single team of experts to have the skills to perform both Cybersecurity and safety tasks simultaneously.” It is for this reason that the Guidebook, while recognizing the advantages of the ideal integrated process, makes provisions for non-integrated safety and security processes that communicate in more or less well-defined ways – what in AQUAS we call interaction points (Figure 1).

We call "interaction point" both an activity and the point in a product life cycle (PLC) at which it occurs. The activity is "interaction" in that (a) experts in the various aspects of the system and its properties interact., e.g. security and safety experts; (b) their analyses are combined in some way, that may be anywhere in the range from informal discussion and mutual critique to using mathematical models to assess various measures of interest for alternative design options, or even a single, summary measure to be optimised (e.g., probability of an undesired event); (c) the need for changes or decisions may be recognised that require an integrated view, e.g. because of inevitable trade-offs between desirable properties, and these trade-offs are discussed between the various experts to produce recommendations/decisions, possibly with the aid of the above-mentioned mathematical models.

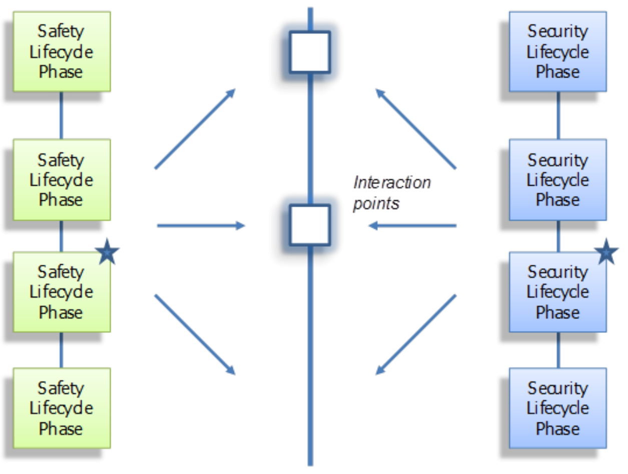

Figure 1 Two separate PLCs with interaction points between them

2 Static versus dynamic interaction points

An important question is when these interactions should take place, to be cost-effective for a given project in a given company. One viewpoint is that the lifecycle model used by the developers should identify from the beginning when interaction points will be needed. These "statically scheduled" interaction points are so scheduled as to achieve a reasonable trade-off between

- The cost of too many interactions for those "lucky" projects that never have conflicts or resulting rework (for these projects, all interactions may be counted in hindsight as unnecessary costs) and

- The cost of too few interactions for the "unlucky" projects, in which conflicts between requirements and unsatisfactory design trade-off are recognised late, requiring expensive rework or causing project failures. For these projects, frequent interaction points would save money by reducing rework.

The standards tend to identify static interaction points, partially through the very nature of the standard as a static text. But the potential improvements through dynamic interaction points are significant. Pre-planned, statically scheduled interaction points are akin to scheduled maintenance of equipment: they happen at predictable times, their cost is factored into the total cost from the beginning, and they are frequent enough to avoid nasty surprises. However, a regime of scheduled maintenance does not necessarily avoid ALL surprises and there is a need to have a design that can deal with failures occurring between maintenance points. If components of a system fail during operation, the system typically needs: means for *failure detection*; means for *diagnosis*; means for *repair* or *reconfiguration, recovery and restart*. In the case of the co-engineered lifecycle, examples of failures and their detection mechanisms might be the following:

- Initial requirements from a client are found to be in conflict during the implementation phase (for instance encryption of data for a particular security standard takes too much time to meet a performance requirement). This may trigger interaction points in the current phase of the PLC, and/or in previous phases (that is, undoing some refinement activity for some system part, going back to change and re-analyse a higher-level design, so as to make the satisfaction of the requirements feasible; or even going back to renegotiate these requirements).
- Inadequate performance may lead to a safety related issue. For example, a machine vision component in an automated system may turn out to be insufficiently robust to adequately recognize a sufficiently large set of risky scenarios and may need to be upgraded for performance. The introduction of new, redundant mechanisms to deliver the needed performance might open up a new attack surface that was previously unanalysed.
- in the process of refining an aspect of design, the design team discovers that they violated some 'contract' established at a previous stage of refinement (e.g., they agreed to implement a certain message encryption as a security control in less than a certain fraction of the main control loop period of a system; but they discover that when implemented it takes longer).
- the safety specialists realise that they may have missed out something important in communicating their proposed architecture to the security team; so, the analysis by the latter that gave the 'all clear' to the architecture may be wrong.
- independently of an on-going development effort (or, alternatively, after deployment), a new vulnerability has been discovered in a component or algorithm. The security team wishes therefore to introduce new controls, which might violate some assumption made by the other teams (e.g. about timing, or about possibility of communication between two components, or authority given to a component) on the basis of the currently specified controls.

In all these cases, the "detection" amounts to some team member becoming aware of something potentially being wrong. Triggering an interaction point (possibly delayed, just as responsive maintenance can be delayed) then serves to perform diagnosis: to decide whether something is indeed wrong, possibly through intermediate steps of more extensive analysis. The interaction point may in turn trigger more extensive analyses (e.g., if our trust that a deadline would not be violated was built simply on extensive statistics of the delays observed in off-line testing, it may trigger another similar round of offline testing), just for the purpose of reaching a diagnosis, and then possibly some rework/redesign, again possibly requiring new analyses on the redesigned system. Analyses of the results of the rework/redesign would be subjected to another interaction point, to check that indeed the problem is resolved. The combination of statically scheduled interaction points and dynamically scheduled ones might prove more cost-effective than a more frequent series of statically scheduled ones.

3 System Design vs. Safety/Security/ Performance Analyses

The evolution of the system through the PLC is captured by models, chosen by the developers. In AQUAS the system models of most of the demonstrators will be based on the OMG SysML/UML formalisms. A significant part of it may be created directly from these models, including by e.g. automatic code generation. Should the system be changed (e.g. fixing faults/vulnerabilities in development or post-deployment), the system model will be modified too, so that the “real system” and the model of it are kept consistent throughout the phases of the PLC.

Assurance about the required non-functional properties of the designed system is achieved by dedicated *methods of analysis* (i.e., Safety, Security, Performance – SSP analysis), focused on assessing whether the system has the required non-functional properties or not. Each one of the various methods used for analysing security, safety and/or performance relies on its *own models*. In some cases, these models coincide with parts of the design documentation: e.g., some verification methods are applied directly to source code or to state machine diagrams used in specifications. But for many SSP analyses, the models they need rely on formalisms that are *very different* from SysML/UML. E.g., performance modelling might use Petri nets or queuing networks. The important point here is that whenever an SSP analysis is needed, a model suitable for it must be extracted or derived from the model of the designed system, available at that particular point in time. Two further important points are worth making here:

- Some methods of analysis (and their respective models) may not be applicable at all before the system model has matured enough (e.g. a tool might need the availability of source code for analysis).
- Some analyses may ignore some details of the designed system even if such details are available. For instance, if one uses a probabilistic state-based model such as Stochastic Petri Nets (SPN) one may be unable to benefit fully from having the full source code of the designed product.
- The design models or design documentation are normally *incomplete* descriptions. For instance, designers may specify the type of a microcontroller or memory chip to use in the system, and so to facilitate verification, appropriate data sheets for these products can be used. But implementation details *inside* these components may have major effects on non-functional properties. E.g., chip mask changes may have undocumented performance implications, or add/remove design faults; the much publicised "Spectre" and "Meltdown" vulnerabilities result from vendor-controlled chip design details that a system designer would typically ignore; and the new security/performance trade-offs required by the fixes for these vulnerabilities were arranged by vendors with limited communication to users. So, analyses for security, safety etc. may require adding extensive "annexes" to system design documentation.

4 Tool Support

Interaction points occur within the context of a number of questions:

- *Why* an interaction point would be needed (e.g. a potential conflict may arise)
- *When* an interaction point should take place (e.g. statically or dynamically determined)
- *What* will take place during the interaction point (e.g. joint examination of a design artefact, trade-off analysis of conflicting design decisions)
- *How* it will take place (e.g. manual observation and discussion, automated tool support, semi-automated tool support)

As challenging as the first two questions are, it is equally challenging to address the second two questions. That is, when an interaction point does occur, there must be a viable set of artefacts (at whatever level of abstraction or lifecycle phase) available.

- *What*. The procedures of e.g. the security and safety analysts can be run independently without difficulty. But they may use different models that are difficult to relate to each other; or, simply, the kind of questions that need to be asked to identity gaps left by the independent analyses are non-obvious. Or e.g. the security analyst may propose a design addition – a subsystem implementing a security control, but specify it in a formalism that makes it hard for the other specialists to analyse. This may create practical difficulties that make a complete analysis too onerous in practice.
- *How*. Even if two artefacts have been created with the same formalism (e.g. SysML), there may be a lack of adequate tools to support the needed analyses (e.g. tools for worst-case performance analysis). More critically, even if the tools are individually available, they may not be able to interact due to poor planning of the overall toolchain framework.

Efficiency of interaction is also an important factor here. People might limit themselves to simpler analyses if it is too time/effort-consuming to do deeper analyses, such as the combined analyses for SSP. Inadequate tool interoperability and inconsistencies of modelling formalisms can severely hamper efficiency, but they can be addressed through emerging interoperability standards. In the end, tool interoperability and judicious automation will improve not only the economics of the work, but also the quality of the result.